

University of Southern California

EE 660 - Machine Learning from Signals: Foundations and Methods

PUBG Finish Final Placement Prediction

Project Type: Individual Collaborative Kaggle Project

Submitted by:

Deepika Kanade

Email Id: kanade@usc.edu,

USC Id: 5369288614

Date: December 6, 2018

Table of contents

1. Abstract.....	3
2. Introduction.....	3
2.1. Problem type, Statement and Goals	3
2.2. Literature Review.....	4
2.3. Prior and Related Work.....	4
2.4. Overview of Approach.....	4
3. Implementation.....	5
3.1. Dataset	5
3.2. Feature Engineering.....	11
3.3. Dataset Methodology.....	17
3.4. Training Procedure.....	18
3.4.1. Models Used.....	18
3.4.2. Parameter Tuning.....	20
3.4.3. Complexity of hypothesis set.....	21
3.5. Model Selection and comparison of results.....	21
4. Final Results and Interpretation.....	25
5. Summary and Conclusion.....	28
6. References.....	28

1. Abstract

The goal of this project is not only to enhance the final performance of a model, but to understand the dataset and explore the variety of algorithms that can be used on it. The PUBG dataset provided by Kaggle has around 4.45 million training data points and 1.92 million testing data points. Post-match statistics of the players in the game are provided, and a Machine Learning model has to be built that will predict the final placement of players between 0 to 1, 0 being the lowest and 1 being the highest.

The project deals with feature engineering to understand the features and perform some modifications to the given dataset. Then the dataset is provided to several models like Linear Regression, Random Forest Regressor, XGBoost and LightGBM. Mean Square Error and R2 scores are used as performance metrics to judge the performance of models and to select the best algorithm and the best model.

Grouping of data points based on matchId and groupId is done and the increase in features by finding the mean, min, max and rank of all the features works the best. LightGBM achieves the best result out of all the models giving a test MAE value of 0.029 and test R2 score of around 0.987.

2. Introduction

2.1. Problem Type, Statement and Goals

Player Unknown's Battle Grounds (PUBG) has enjoyed massive popularity in the past few years. In this game, upto 100 players are dropped in an island empty-handed and they have to scavenge, explore and eliminate other players until one person is left standing. The challenge in this game is that the size of the island keeps shrinking after a certain duration of time.

The PUBG dataset available on Kaggle has upto 100 players in each match and the players can form teams in a particular match. The dataset provided has approximately 4.45 million data points and 29 features where each row reflects a person's post-match statistics. The goal of the project is to get the resulting prediction to be in the range of 0 to 1, 1 being the first place and 0 being the last place in the prediction.

The goal of this project is also to explore the existing features, add some new features in the system and experiment with different models to choose the best performing model. This is a **regression problem** where the target has continuous values in the range of 0 to 1. The variable to be predicted is 'winPlacePerc' indicating the quantile percentage of placement for each player.

The project is extremely interesting as it involves handling a huge dataset, understanding features, performing Exploratory Data Analysis (EDA) to decide which features to drop or add.

The project was challenging due to the following reasons:

- Number of data points is extremely large compared to the number of features. Hence, the problem of underfitting was prominent.
- Running models on such a huge dataset is a task. Also, cross validating the features becomes cumbersome.
- There was a non-linearity in the dataset which had to be dealt with by using non-linear regression models.
- Significant amount of feature engineering was required.

2.2. Literature Review

As this is an ongoing Kaggle Project and comprises of a real-world dataset, there were no publications in this regard. Literature Review comprised of exploring the Kaggle collaborative forum and getting ideas from other people in the project competition.

2.3. Prior and Related Work – None

2.4. Overview of Approach

As I mentioned earlier, the main aspect of this project was exploring features and deciding which features to retain or drop. My analysis and experimentation revolved around this.

I can summarize the experiments I did in the following way:

- Using all the features and label Encoding the only categorical feature present.
- Using all the features and One Hot Encoding the only categorical feature present thereby increasing the feature dimension.
- Using Principal Component Analysis to reduce the number of features on the basis of large inter-correlation between features.
- Subsampling the data to reduce the computation of training. This was further carried out in 2 ways:
 - 1) Outlier removal feature-by-feature
 - 2) Random sampling of data
- Manually selecting features based on Heat Map.
- Adding features by performing non-linear combinations of existing features.
- Grouping data points by matchId and groupId and aggregating the values to reduce number of data points and increase the number of features.

The performance metric used to judge by performance was Mean Absolute Error (MAE) and R2 score.

3. Implementation

3.1. Dataset

The PUBG dataset available on Kaggle has upto 100 players in each match which are uniquely identified based on their matchId. The players can also form teams in a match by having the same groupId and the same final placement in the match.

The data comes from different groupings between the player and hence, there is no guarantee that 100 players will be there in a match or not more than 4 people in a group. The teams can be formed based on solo, duo, quad and customs.

There is a total of 4.45 million train data points and a total of 1.93 test data points. There are in all 29 features.

The input features can be summarized as follows:

Sr.no	Name of feature	Type of Feature	Brief Description
1	Id	String	Unique Id for each person
2	matchId	String	MatchId to indicate people in a match
3	groupId	String	GroupId to indicate people in a group
4	assists	Real	Assisting team mates in killing
5	boosts	Real	Number of boosts items used
6	damageDealt	Real	Total damage dealt
7	DBNOs	Real	Number of enemy players knocked
8	headshotKills	Real	Number of Players killed with headshots
9	heals	Real	Number of healing items used.
10	killPlace	Real	Ranking based on number of players killed
11	KillPoints	Real	Ranking based on kills
12	Kills	Real	Number of enemy players killed
13	KillStreaks	Real	Number of enemy players killed in a short amount of time
14	longestKill	Real	Longest distance between player and player killed at the time of death
15	matchDuration	Real	Duration of match in seconds
16	maxPlace	Real	Maximum place based on data in the match
17	numGroups	Real	Number of groups for which data is provided in the match
18	rankPoints	Real	Ranking of a player

19	revives	Real	Number of times the player revived his teammates
20	rideDistance	Real	Distance travelled in meters
21	roadKills	Real	Number of kills done while in vehicle
22	swimDistance	Real	Total distance travelled swimming
23	teamKills	Real	Number of times a player killed teammate
24	vehicleDestroys	Real	Vehicles destroyed
25	walkDistance	Real	Total distance travelled walking
26	weaponsAcquired	Real	Number of weapons picked up in the game
27	winPoints	Real	Win-based ranking of player
28	matchType	Categorical	Game-mode used by player in match
29	winPlacePerc	Real	The target of prediction

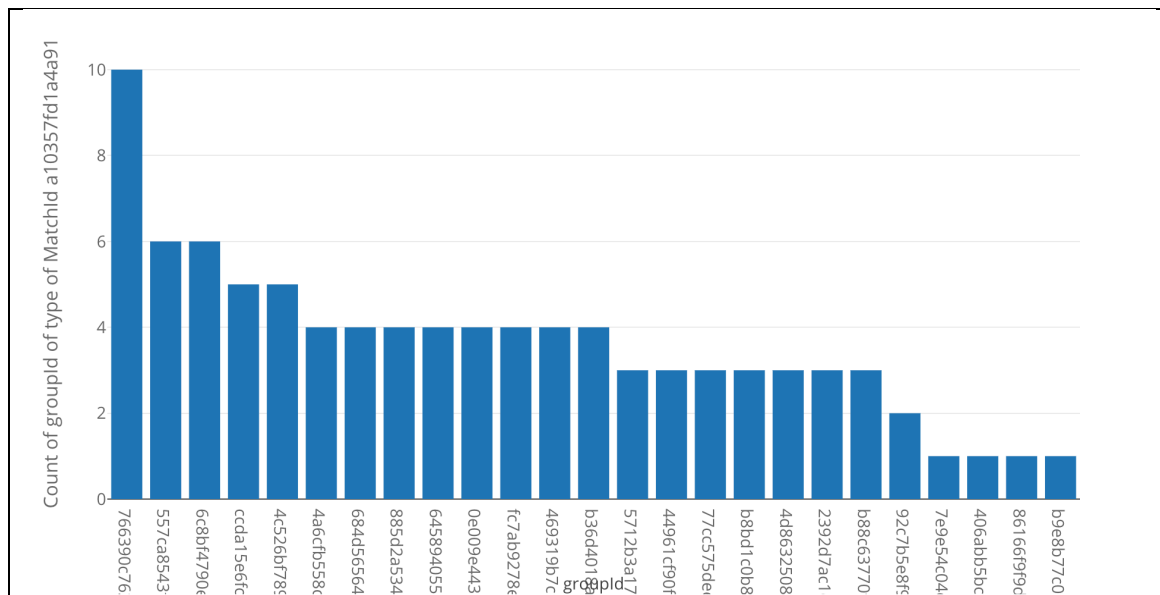
Table 1: Feature Description

I will explain the important features based on my analysis on a pre-training dataset from the entire training set. My pre-training set is 10% of the entire dataset which is around 400,000 data points. Also, I have done preliminary analysis by splitting data based on matchId.

I) GroupId and MatchId

MatchId is used to identify players in the same match. There can be at the most 100 players in each match.

GroupId is used to identify players in a group for a match. If same players play in different matches, they will have different groupId in every match.

**Figure 1:** Histogram of matchId and groupId

I have filtered players based on the matchId and there are total 91 players in this match. But as it can be seen from the plot, the maximum number of players in the game having same groupId is 10 which is due to some players exiting the game and also due to the custom matchType.

II) Assists and Kills

Assists deal with how many people have assisted killing people even though they haven't killed anyone on their own. Assists and kills are related as in kills, they are recorded based on how many people a person kills.

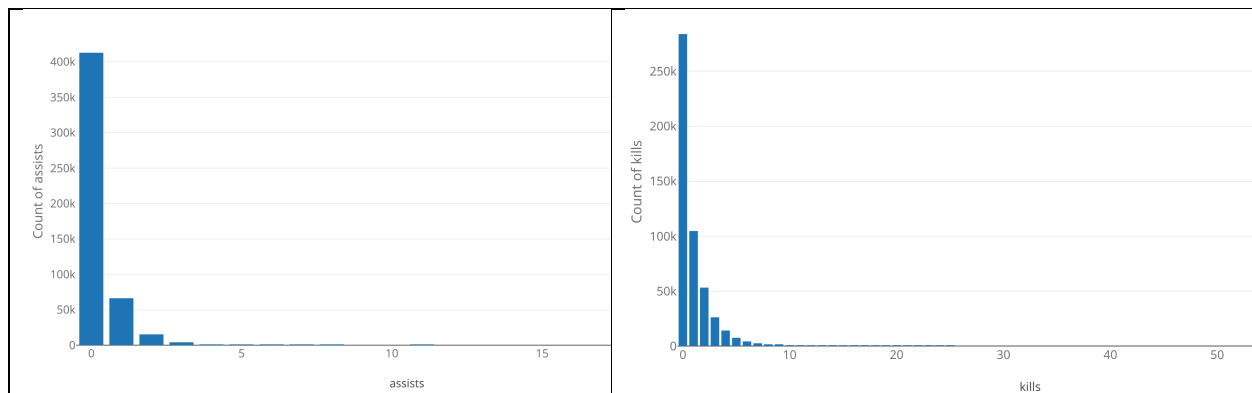


Figure 2: Histogram of Assists and Kills

As it can be seen, in both assists and kills, the count of zeros is high. But these features are still important while determining the final rank.

III) KillStreaks

It indicates the maximum number of players killed in a short duration of time.

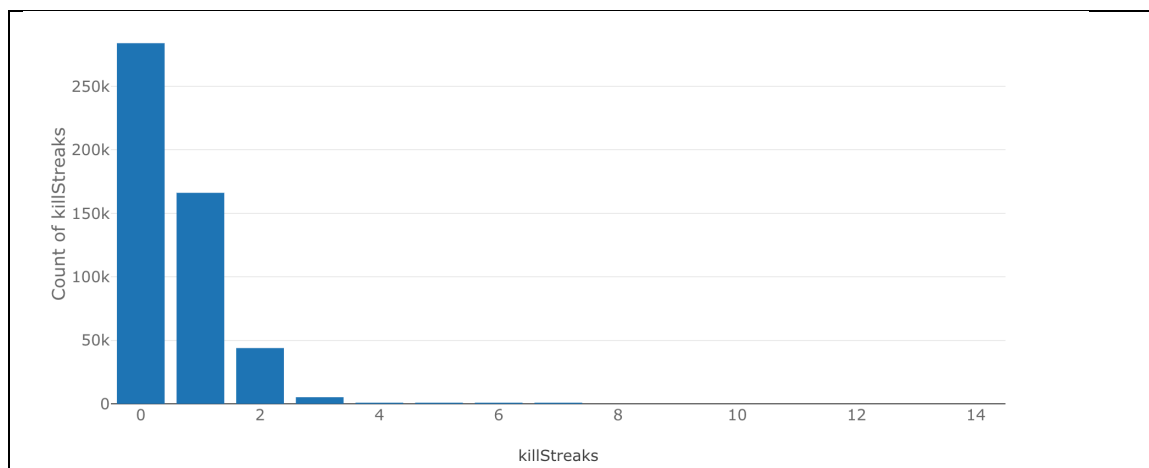


Figure 3: Histogram of KillStreaks

As it can be seen from the figure, some players kill a lot of people indicating that this can be crucial while determining the rank of a player.

IV) RoadKills and TeamKills

Roadkills indicate the number of people killed while travelling in a vehicle whereas TeamKills indicate the number of people killed by a team member within the same team. These features seem to be useless as it is highly unlikely that this will happen which can be proven from the figures below.

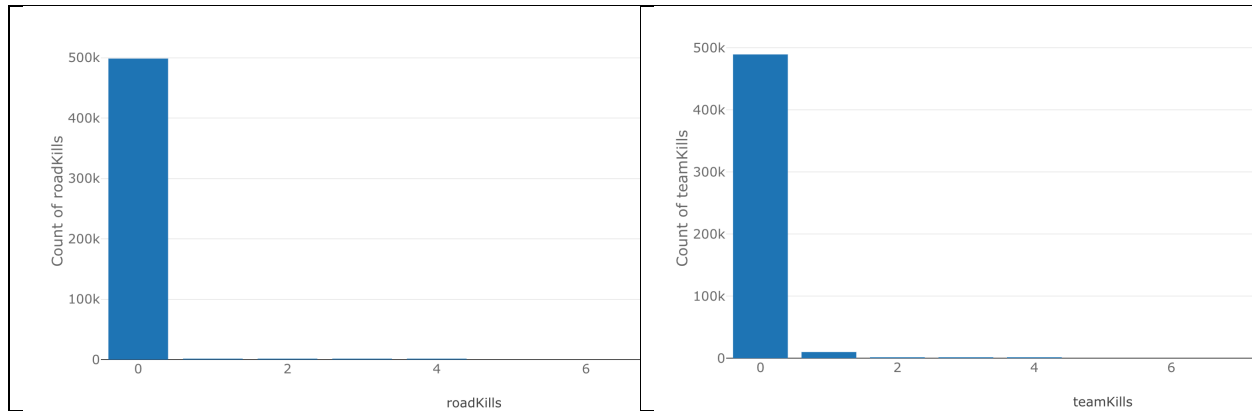


Figure 4: Histogram of roadKills and teamKills

V) HeadshotKills and DBNO's

Headshot kills indicate how many players could aim at the head of enemy and kill the enemy right way. DBNO's indicate number of players knocked but not killed. These are very important features to judge the placement percent of that player.

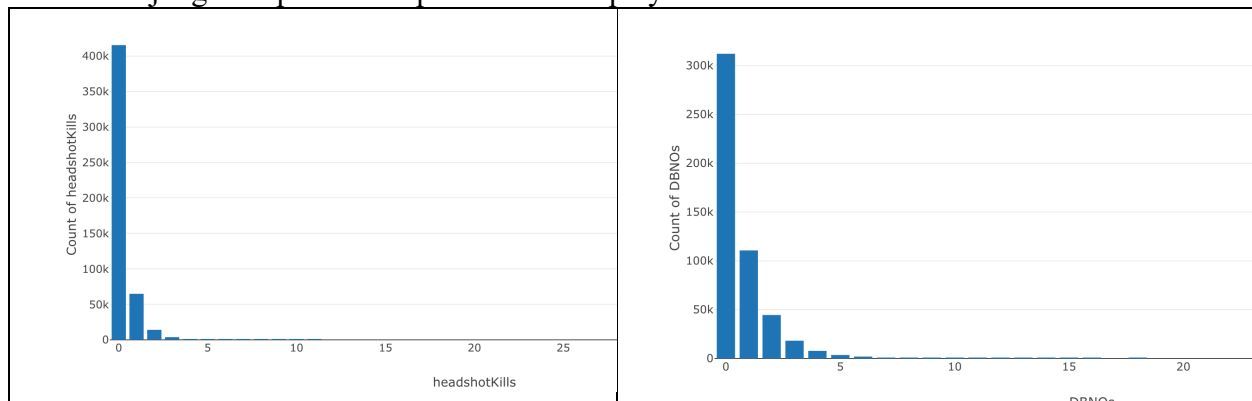
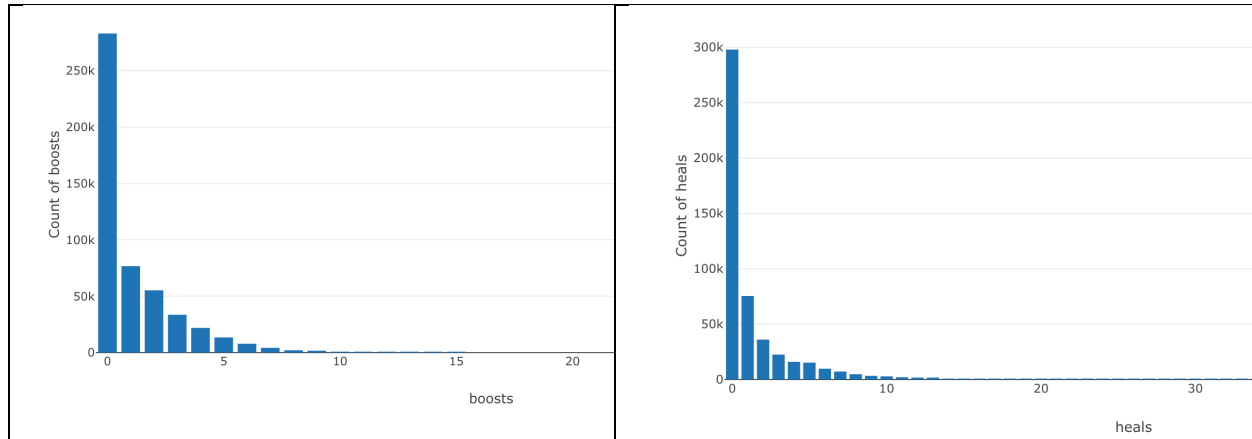


Figure 5: Histogram of headshotKills and DBNOs

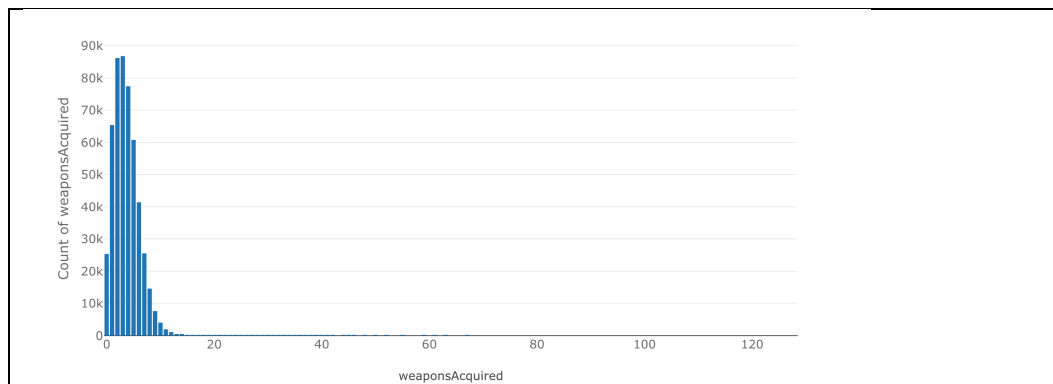
VI) Boost and Heal

Boosts and Heals are used to revive the person in the game. Boosts have an immediate effect whereas heals take a longer time to revival. However, both are important to judge the time a person will be in the game.

**Figure 6: Histogram of boosts and heals**

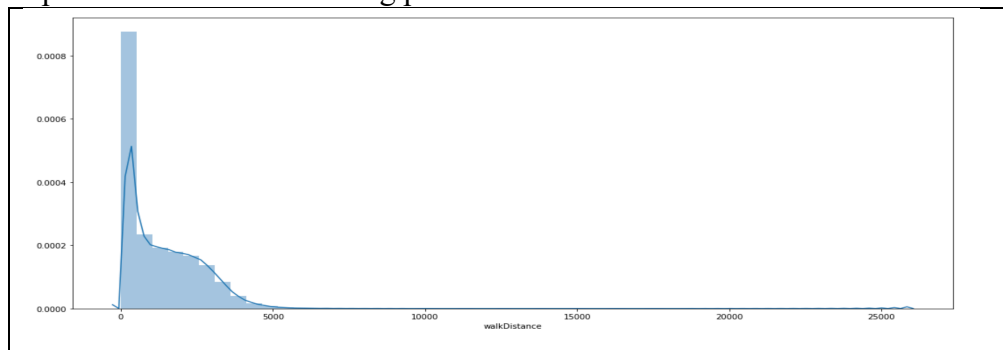
VII) Weapons Acquired

This feature indicates how many weapons a player has acquired in the game. As seen in the figure, a lot of players acquire a lot of weapons which will help them stay in the game longer.

**Figure 7: Histogram of weaponsAcquired**

VIII) Walk Distance

This indicates the maximum distance a person travelled on foot. Larger value of walkDistance indicates that the person was in the game active for a long time. Hence, this is one of the most important features influencing prediction.

**Figure 8: Histogram of walkDistance**

IX) MatchType

The only categorical feature present was matchType which had 16 unique values which are shown as below:

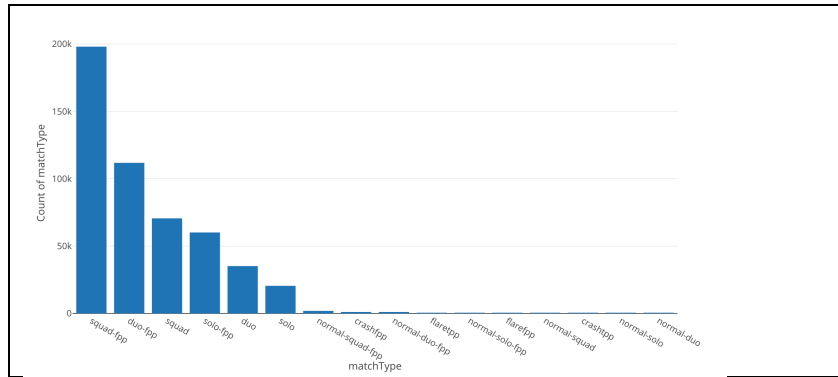


Figure 9: MatchType feature analysis

3.2. Feature Engineering

As I have mentioned earlier, the main aspect of this project was data analysis and feature engineering. I experimented with a lot of pre-processing techniques which can be summarized as follows:

I) Pre-processing of continuous and real feature values

As a lot of features had continuous values, I performed standardization of features i.e. making all features have 0 mean and unit variance. For this purpose, I used the `sklearn.preprocessing.StandardScaler()` function and fit this model on training data, transforming the training and testing data alike.

I also experimented using the only categorical Feature by one hot encoding it and label encoding it. Label Encoding gives unique value to each category and replaces the categorical string by the numerical value in the column itself. One Hot encoding however, creates new columns depending on the number of unique categories wherein each column has binary values. I used the `sklearn.preprocessing.LabelEncoder()` and `sklearn.preprocessing.OneHotEncoder()` for this purpose.

II) Replacing/ Removing missing values in the data

The data had a missing value in the target column which I replaced with a 0. I used the `np.fillna()` function for that. Also, there was one data point which had a lot of Nan values in a lot of columns, so I dropped that data point.

Also, as I created non-linear features to increase the feature size, I divided feature values. So I obtained a lot of Nan values which had to be replaced using the `np.fillna()` function.

III) Principal Component Analysis (PCA)

After exploring the features, I tried to look into the heat Map of all these features. The heat map depicts the correlation between different features as well as correlation of the features with the target.

The heat map was obtained on the pre-training dataset and is depicted as follows:

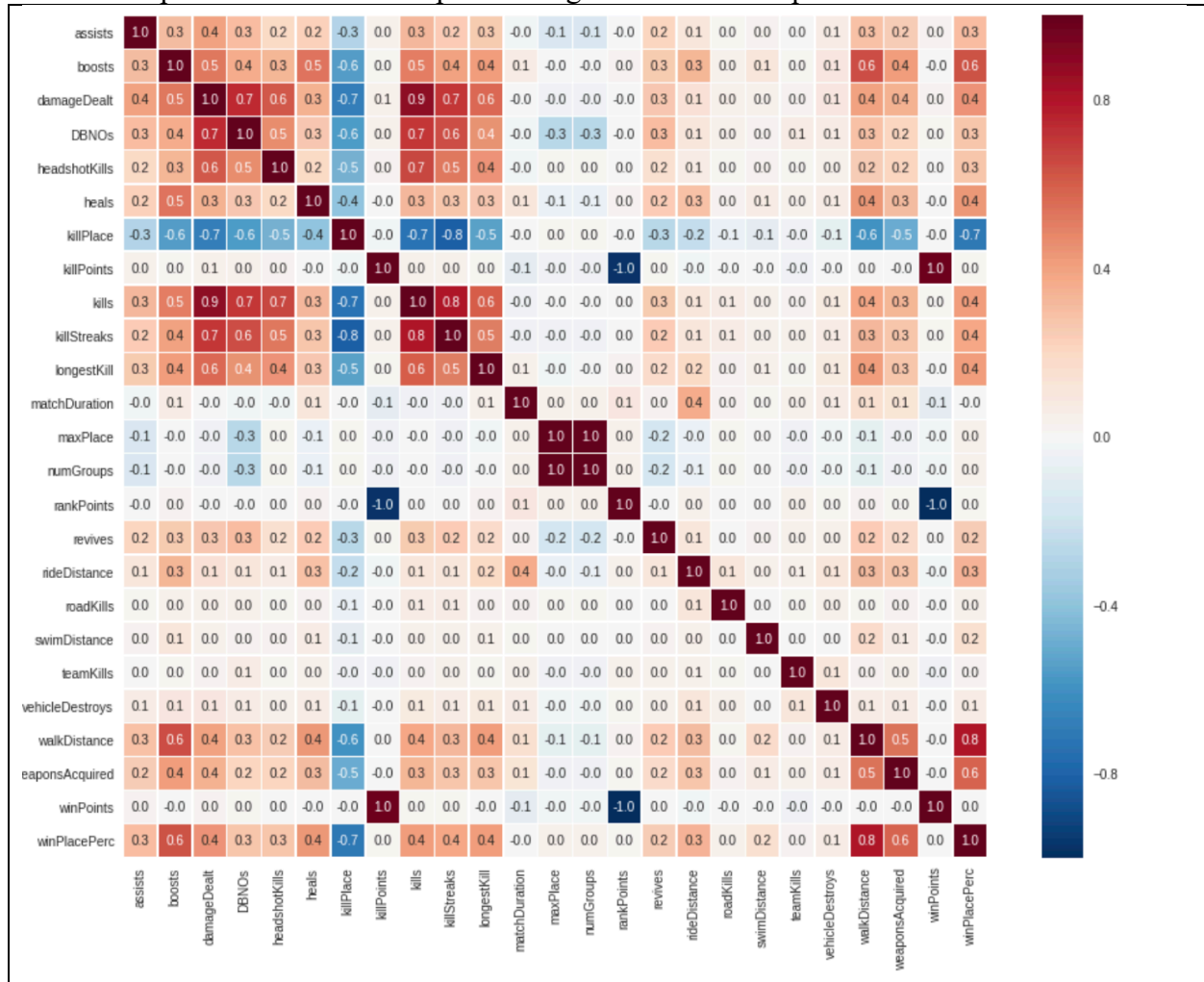


Figure 10: Heat Map of all features

As it can be seen in the Heat Map, a lot of features have correlation between themselves. This instinctively gave me the idea to try PCA to reduce the dimensions and also reduce the redundant features in the system by projecting features on a lower dimensional subspace.

PCA is a dimensionality reduction algorithm which retains the direction of maximum variation of data and projects the data on a lower dimensional space. It can be implemented by forming a covariance matrix P of all the data points X ,

$$P = XX^T$$

And then finding the eigen decomposition,

$$P = Q\Lambda Q^{-1}$$

where Q is the square matrix whose ith column is the eigenvector q_i of P and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues. These eigen values are then arranged in descending order to get the eigen values capturing the maximum variance of data and the reduced dataset is formed.

I implemented PCA using a standard library function `sklearn.decomposition.PCA` and gave a parameter retaining 95% of the variance in the data. The number of components used to represent 95% variance in the data were 5. Hence, the data was projected on a 5 dimensional subspace.

This gave me a comparative high MAE score obviously as the number of data points is anyways larger than the number of dimensions. So, decreasing the features further led me into underfitting. But I wanted to try this approach to analyze the results.

IV) Manually selecting features based on Heat Map

I also manually tried deleting the following features as they had a high correlation between themselves and low correlation between output which can be inferred from the Heat Map.

I dropped the following features:

- ID
- matchId
- groupId
- rankPoints
- winPoints
- killPoints
- maxPlace, numGroups
- roadKills
- teamKills
- matchDuration

But as expected, the final MAE score did not change much after dropping these features.

V) Adding new features

As I tried experimenting earlier with dropping features manually or using PCA earlier, which did not help, I tried adding new features in the system based on my knowledge of the game as well as the inter correlations between the features.

The following features were added:

1) *Head Shot Rate*

$$\text{HeadShot rate} = \frac{\text{Kills}}{\text{Headshot Kills}}$$

2) *KillStreak Rate*

$$\text{KillStreak Rate} = \frac{\text{Kill Streaks}}{\text{Kills}}$$

3) *Distance*

$$\text{Distance} = \frac{\text{walkDistance} + 0.4 \text{ RideDistance} + \text{swimDistance}}{\text{matchDuration}}$$

4) *Health items*

$$\text{Health Items} = \frac{\text{Heals}}{\text{Boosts}}$$

5) *Boosts Ratio*

$$\text{Boosts Ratio} = \frac{\text{Boosts}^2}{\sqrt{\text{walkDistance}}}$$

After finding the correlation of these feature with the target, they had a high correlation indicating these will be a good features for learning.

VI) **Grouping features based on GroupId and MatchId**

According to the data provided by Kaggle, in a match, people with same groupId form a group and that group has the same target placement in that match. This was according to me one of the main challenges the model faced as for the same target value, it had different feature values, leading to confusion for model learning. So, to alleviate that, I decided to group the data points based on groupId and matchId and aggregate their feature values to be represented as one row for each group in the match. This lead to decrease in the data set as well. I also ranked the features after finding the mean based on their matchId.

This process increased my features to 162 as every feature had been expanded to contain it's mean, min, max value as well as rank was decided based on these values. Also, due to the grouping my data points reduced to about 2 million in the train set.

I have summarized my method in the flow chart below.



Figure 11: Flow chart for grouping on train and test data separately

VII) Subsampling of data

As the dataset provided on Kaggle has a lot of data points, I tried to subsample the data points and run models with these reduced data points. To subsample the data, I tried two methods namely Outlier Removal and Random sampling the data.

A) Outlier Removal

After observing the boxplots of each of the feature, I deduced that a lot of features had outliers within them. Instead of randomly removing data points and training a model on that, I decided to remove the outliers in the data.

The box plots for the features are as shown below:

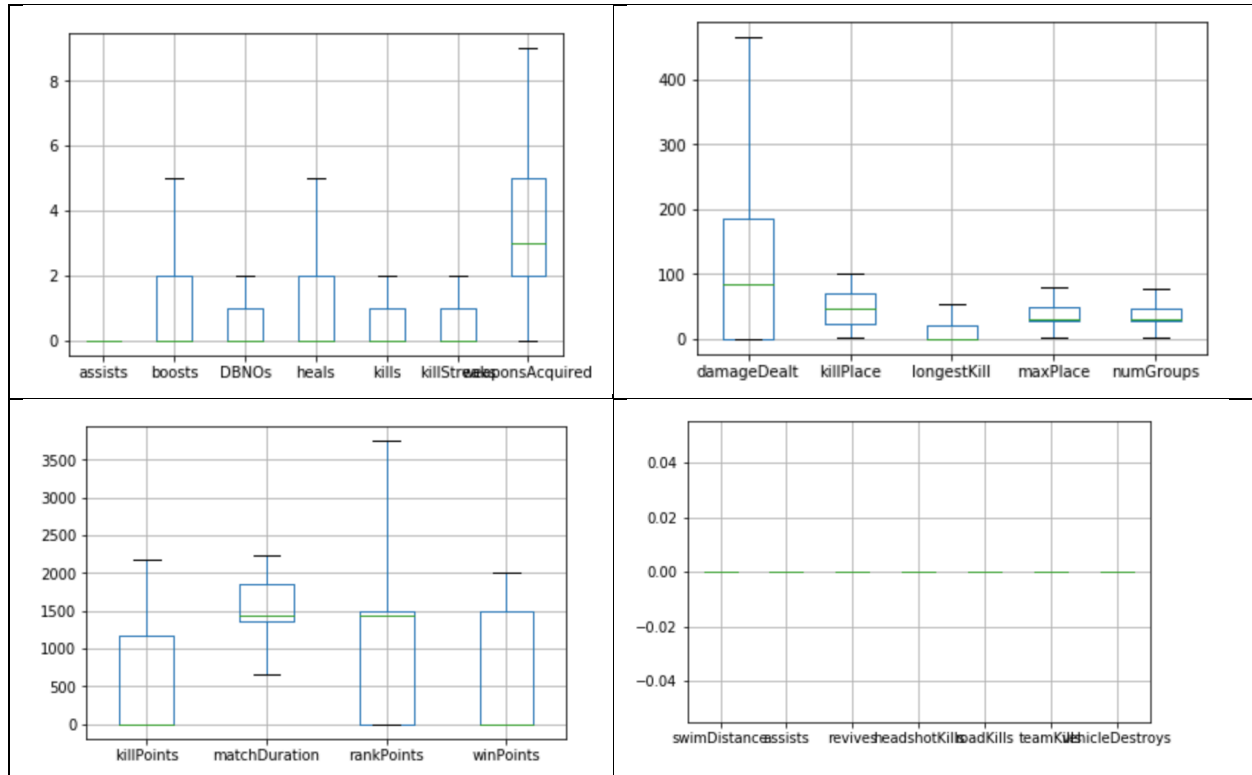


Figure 12: Box plots for each feature

As seen from the above figures, some features had outliers in them which were removed by setting upper and a lower threshold values. Some features which did not have outliers were retained as it is.

Based on the outliers, the following upper and lower threshold values were set for every feature. The data points having values lesser than the lower threshold and higher than the upper threshold were then dropped. This reduced the dataset size.

Feature	Lower Threshold	Higher Threshold
Boosts	0	5
<u>damageDealt</u>	0	465.25
DBNO's	0	2.5
heals	0	5
killPlace	0	141.5
killPoints	0	2927.5
kills	0	2.5
killStreaks	0	2.5
<u>longestKills</u>	0	53.324
<u>matchDuration</u>	641	2577
maxPlace	0	80.5
numGroups	0	77
<u>rankPoints</u>	0	3751.5
<u>rideDistance</u>	0	0.510875
<u>walkDistance</u>	0	4709.85
<u>weaponsAcquired</u>	0	9.5
winPoints	0	3737.5

The outlier removal process had to be done on the test data as well, as the test data was very large compared to the train data after outlier removal.

B) Random Sampling of data

To reduce the computation for the model, I tried randomly selecting a sample of the data and training a model based on that. But as expected, the result was not very good as the sub sampled data was not representative of the entire dataset. The match and group characteristics of the data were not retained.

3.3. Dataset Methodology

The Kaggle competition has a train and test set which is provided. However, the test set has no target values, so the performance of the system can't be tested on the test data. Hence, I had to split my train data in train and test data.

80/20 split was used to split the train data into pseudo train and pseudo test data.

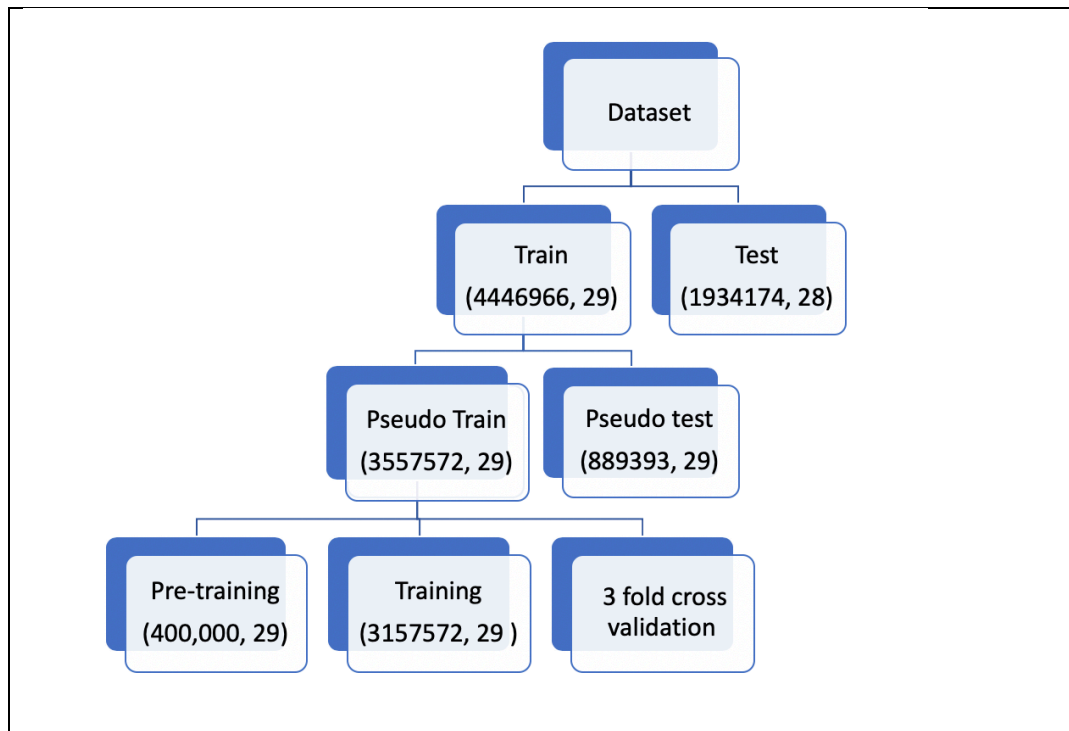


Figure 13: Dataset usage

As the dataset was extremely large, running cross validation loops every time was cumbersome. Hence, I ran 3-fold cross validation for my baseline model using all the features and obtained the best parameters and then used these parameters for all the models I used later.

The following things were observed in the dataset usage:

- 1) All the initial analysis was done on the pretraining dataset. So, the assumptions of machine learning were not violated. I also removed the pretraining set from the original training set.
- 2) 3- fold Cross validation was performed just before training a model. I did not have a separate validation set. The validation results were used to decide the best parameters for a model that will give the lowest error.
- 3) Test set was untouched till the prediction was made. It was used 4 times for 4 different models and all my decisions about the performance of the model were made on the test set.

3.4. Training Process

3.4.1. Models used

I used 4 machine learning models to train on the dataset and chose the best model based on the MAE value of each model.

1) Linear Regression

I tried Linear Regression as my baseline model. As it is a simple model, comparisons can be made with respect to this model. Linear Regression is a statistical method to predict the relationship between an independent variable and a dependent variable. This problem dealt with the prediction of a predictor variable. In Linear Regression, the unknown function that maps the dependent variable to the independent variable has its model parameters estimated from the data. After fitting the linear Regression model, if additional data is provided to the model, it predicts the predictor variable automatically.

The model assumes to have a linear relationship in the following way,

$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i = 1, \dots, n,$$

This is then solved using an ordinary least square solution wherein the parameters of the model are chosen to minimize the least square values between the predicted and the actual value of the predictor variable which is given as follows:

$$S(\boldsymbol{\beta}) = \sum_{i=1}^m |y_i - \sum_{j=1}^n X_{ij} \beta_j|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2.$$

I used the `sklearn.linear_model.LinearRegression()` function to implement this model.

2) Ridge and Lasso Regression

Ridge and Lasso are an extension of the Ordinary Linear Regression wherein a regularizer term is added. The regularizer term is used to penalize the higher order weights and to increase the sparsity of weights in the model. Regularizer is used in the case of overfitting and the amount of regularization to be added can be decided. A prior term is added when using Ridge or Lasso Regression wherein the prior term for Lasso is Laplacian and for Ridge is Gaussian. In the given dataset, chances of overfitting were very low as the number of data points were extremely high compared to the number of features, but I still wanted to see if that MAE value changes after using a regularizer. The MAE values were exactly the same as that of the Ordinary Least Square solution indicating that the use of regularizer is not needed.

I used the `sklearn.linear_model.Ridge()` and `sklearn.linear_model.Lasso()` functions to implement these models.

3) Random Forest

Random Forest is one of the main models used for predictive modelling as it uses the ensemble model approach. As it is a non-linear model, I wanted to try this on my dataset and as expected the loss reduced after using Random forest. Random Forest as an ensemble model as multiple decision trees are built during training. During testing, the average of the decisions from multiple trees is taken and assigned to be the final predicted value. Random Forest is a strong learner which combines multiple Decision Trees i.e. weak learners to build the system. Random forest works by randomly sampling multiple subsets from the whole dataset with replacement. This is called as bagging. Due to this, the variance of the final model is reduced in turn leading to a consistent estimator. The final prediction for regression is given as follows:

$$\mathbb{P}(\hat{y} = c | x, D^*) = \frac{1}{B} \sum_{b=1}^B \mathbb{P}_c^b(x)$$

The equation indicates that the average of the predictions from all the trees is taken for assigning the final prediction.

I used the `sklearn.ensemble.RandomForestRegressor()` function to implement this model.

4) Light GBM

Light Gradient Boosting Method (LightGBM) is a gradient boosting method that uses a tree-based algorithm. Gradient Boosting is a method where weak learners are added to build a strong learner using gradient based approaches. The specialty of LightGBM is that it is a leaf-based algorithm compared to all other approaches which are level-based. In this method, the tree is grown on leaves and hence as the depth of the tree increases, the complexity of the model increases.

However, for large datasets LightGBM is extremely popular as it runs on high speed with large datasets and also required lower memory to run. It focuses on decreasing the final accuracy thereby growing the tree on the leaf with maximum delta loss. It also supports GPU learning. For

smaller datasets, it might lead to overfitting but as the dataset I have used is very large, it works the best. However, as a lot of parameters are present, hyperparameter tuning is a bit cumbersome.

I used the following command, `import lightgbm as lgb`, to import the lightGBM module. LightGBM works extremely well for large datasets so I tested on this model.

5) *XGBoost Regressor*

As mentioned earlier, decision trees are considered to be weak learners. Hence, strong learners are developed from these weak learners. XGBoost is the abbreviation for eXtreme Boosting. This also uses the Gradient Boosting Decision Tree algorithm. Gradient Boosting is an approach where new models are added to the existing models to decrease the loss and the combined result from all these models is used as the final prediction. It uses the gradient descent algorithm to minimize the loss when adding new models. The execution time of the XGboost model is extremely small and it also uses the leaf-based tree growing. XGBoost is a very popular model used in Kaggle competitions due to its ability to handle large datasets.

XGBoost works extremely well for large datasets so I used this model in my analysis.

3.4.2. Parameter tuning

As the dataset was large, running cross-validation to choose the best parameters for a model for every different algorithm I tried was a task. Hence, I performed a 3-fold cross validation using GridSearchCV method on the basic model wherein I used all the features. I retained these model parameters for the other algorithms I tried. I used the function `sklearn.grid_search.GridSearchCV()` to do this task. I used hyperparameter tuning for all the 4 models namely Linear Regression, Random Forest, XGBoost and LightGBM. I used GridSearchCV to get the parameters that gave the lowest MAE score. I then used these parameters to retrain the model with the training data.

The parameters selected after cross validation are listed below:

Model	Parameters
Linear Regression	"Fit intercept : True"
Random Forest	"n_estimators=40, min_samples_leaf=3, max_features='sqrt', n_jobs=-1"
XGBoost	"n_estimators=1650, max_depth=9, learning_rate=0.007, num_iterations = 600"
LightGBM	"objective" : "regression", "metric" : "mae", 'n_estimators':20000, 'early_stopping_rounds':200, "num_leaves" : 31, "learning_rate" : 0.05, "bagging_fraction" : 0.7, "bagging_seed" : 0, "num_threads" : 4, "colsample_bytree" : 0.7

Table 2: Parameters chosen for each model

3.4.3. Complexity of Hypothesis set

The total number of data points provided in the training set were around 4.45 million and those in the test set were 1.93 million. The feature dimension provided was 29. Hence, clearly the number of data points were very high compared to the number of feature dimensions. Hence, the problem of overfitting was eliminated. I still tried to incorporate Regularizer in the form of Ridge and Lasso Regression. However, as expected the results did not improve as there was no overfitting in the first place.

However, according to me the model faced the problem of underfitting as the input data points were too high compared to the feature dimensions. Hence, to curb this problem, I decided to incorporate more manual features in the system and also tried to reduce the data points by using the method of grouping.

3.5. Model Selection and Comparison of Results

As I had multiple models in the beginning namely Linear Regression, Random Forest Regressor, XGBoost Regression and LightGBM, to select the best model, I used the metric Mean Absolute Error (MAE) and R2 score. These metrics have been explained below:

1) Mean Absolute Error (MAE):

Mean Absolute Error is the measure of difference between two continuous variables. For this problem, I had actual values of the target and the predicted values of the target. To calculate the loss in general for a model and to evaluate its performance, I calculated the difference between the actual and the predicted values.

$$\text{MAE} = \frac{\sum |\text{actual target value} - \text{Predicted target value}|}{\text{Number of data points}}$$

2) R2 score:

The R2 score is used as a performance measure to judge the performance of the predicted value. The best value of R2 score is 1 and its value can go in negative as well. A value of 0 indicates that the model is predicting the aggregate value of the predictor variable.

As mentioned earlier, the test dataset provided by Kaggle is not of any use for evaluating performance of the model as it does not have the actual target values. Hence, the results mentioned are on **pseudo test data split from the train data** provided by Kaggle.

As mentioned in section 2, I tried multiple approaches. I have consolidated the results from all the approaches below:

I) Using all the features in the data and One Hot Encoding the only categorical feature 'matchType' and testing on all Models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.08990819996333732	0.08986666440532054	0.8397285519609797	0.8395396067420036
Ridge Regression	0.08990557751874433	0.08986428156248807	0.8397294610506324	0.8395418992029918
Lasso Regression	0.09245110171794275	0.0924042927438145	0.8304844207547856	0.8303366967780034
Random Forest Regressor	0.02285087735916281	0.06033188288701576	0.9886178447565763	0.9237874028968516
XGBoost Regressor	0.053246452719138	0.061256472718193	0.874637328429392	0.873626327272828
LightGBM	0.054346632238434	0.056030832456534	0.974838492394944	0.935854949495443

As Lasso and Ridge gave same results as that of linear regression, I decided to drop them in the further analysis.

II) Using Principal Component Analysis and testing on all Models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.1048869206362592	0.104817580	0.7925364050634163	0.792746549465186
Random Forest Regressor	0.038687983604746935	0.07001214770126958	0.9663359067179127	0.8966215231318947
XGBoost Regressor	0.0832733832939	0.081256472718193	0.829393039329323	0.812784738738248
LightGBM	0.054346621323232	0.05603083565554	0.872839849384	0.883728473738472

III) Outlier removal on all features to decrease training data size and testing on all models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.07729700421207321	0.07728260763928234	0.8240988569990519	0.8240313591827706
Random Forest Regressor	0.06898598495893485	0.061287384738474	0.913898483928494	0.907456573645763
XGBoost Regressor	0.689409405930950	0.07019854894385	0.90384938494344	0.894756473984394
LightGBM	0.04939388237994846	0.05151000781779391	0.9299767314896638	0.9208841335693773

IV) Manually selecting features based on Heat Map and testing on all models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.09786744199893967	0.0977935973029451	0.8091267940493925	0.808971669829127
Random Forest Regressor	0.020943062208365854	0.055310058405587115	0.9862437680551359	0.908163688763320
XGBoost Regressor	0.043849834983434	0.05440543543543	0.95635463564563	0.913892838928823
LightGBM	0.06668050063613902	0.06756034216399925	0.9090973842725627	0.905193788838131

V) Adding new manual features with existing features and testing on all models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.08356236268314907	0.09075851814048325	0.8605930098950516	0.832130738330178
Random Forest Regressor	0.03613276324775616	0.06335083320354007	0.9716439437780066	0.90969308587117

XGBoost Regressor	0.044583485592094	0.06473847827483744	0.957874857837845	0.912839489328984
LightGBM	0.04600113434334	0.056022745656565	0.98348329849384324	0.92387483748738

VI) Grouping data points by matchId and groupId and finding new features by finding max, min, mean and rank of the features along with adding new manual features and testing on all models:

Models	Train MAE	Test MAE	Train R2 score	Test R2 score
Linear Regression	0.049656894208972964	0.04959177804577209	0.9491755583509244	0.949246948427174
Random Forest Regressor	0.018935538574706784	0.036777733387011693	0.9910678567107042	0.96859571874025
XGBoost Regressor	0.034898938493849	0.3578457487585455	0.994857843758475	0.95485438758475
LightGBM	0.023675767672346	0.02925782343434	0.9883748374874	0.98745847583748

The final algorithm I selected was the last one wherein I added 5 features manually(referred to in section 3.2), and I grouped features by their groupId and matchId and found mean, min, max and rank of all the features. This led to an increase in dataset size. The best model was **LightGBM as it gave the best R2 score and the least MAE value.**

As the dataset was very large, plotting a 3D plot was very difficult. I have plotted the predicted value of the target with the actual value of the target.

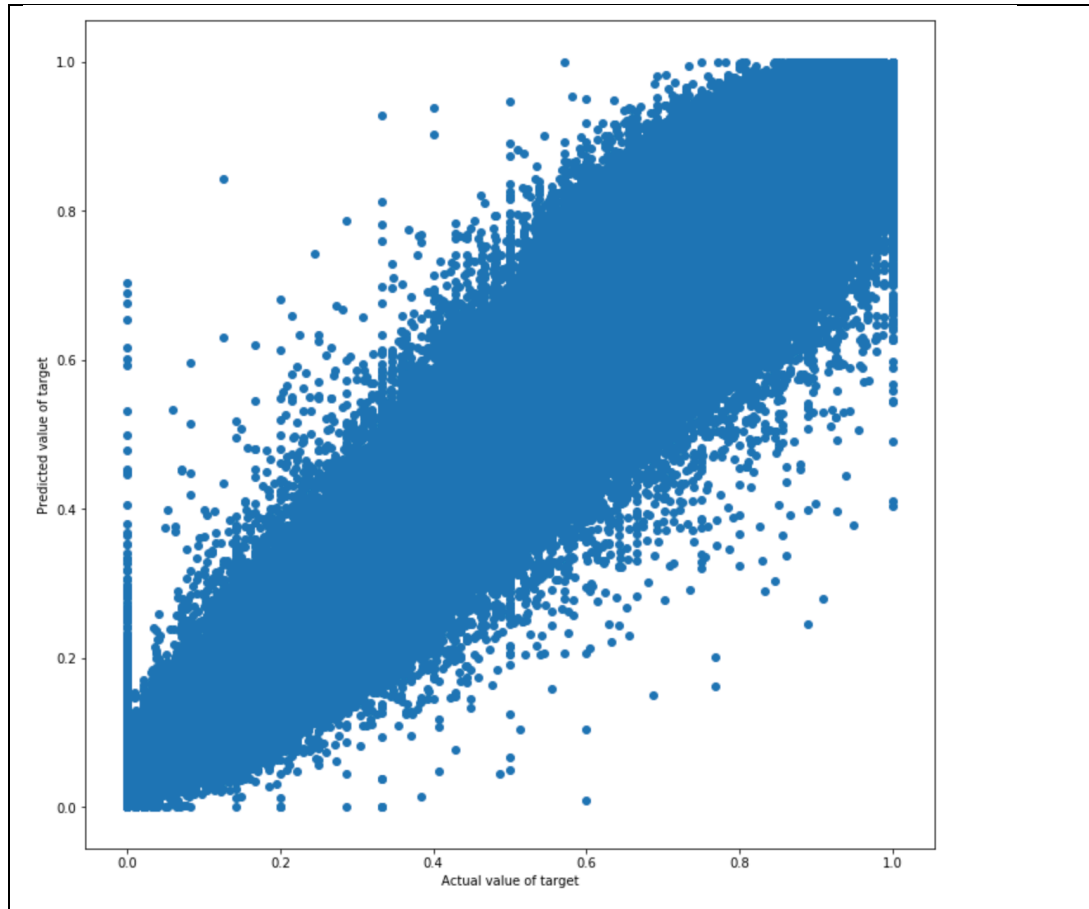


Figure 14: 2D plot of resulting regression function against the target function

According to figure 14, my predicted and actual values seem to be almost equal as I can see a scatter plot centered at the 45 degree line. This proves that the model is doing a decent job for prediction.

4. Final Results and Interpretation

As mentioned earlier, I tried a variety of algorithms and tested these algorithms on different models to see the best performing model. To judge the best performing model, I used the metric Mean Absolute Error (MAE). The best performing model was **LightGBM and the method of grouping along with new features** added helped.

I have mentioned the flow chart of the best performing model and algorithm below:

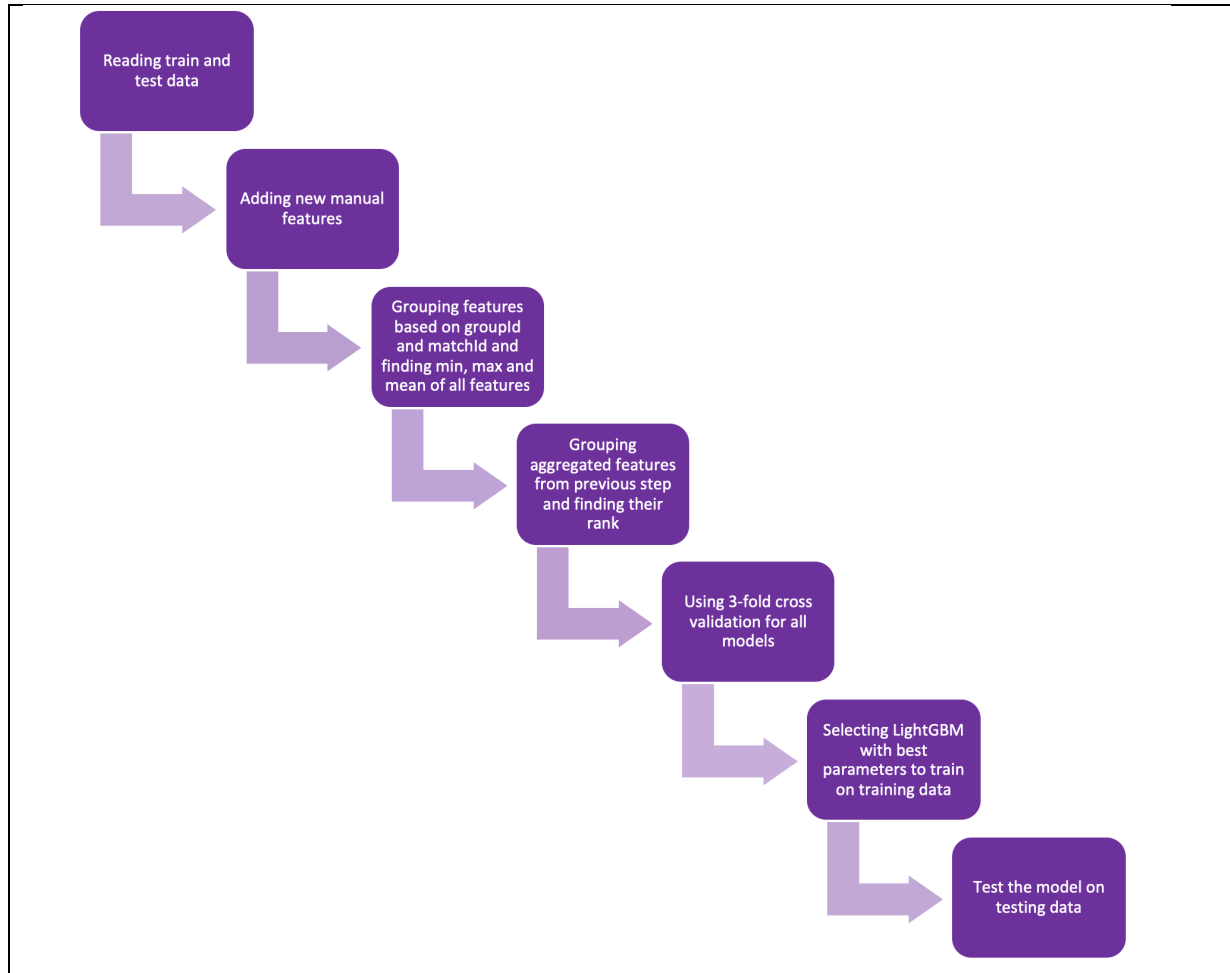


Figure 14: Algorithm for best algorithm and model


The parameters of the best model are listed as follows:

```
Params = { "objective" : "regression", "metric" : "mae",
           'n_estimators':20000, 'early_stopping_rounds':200,
           "num_leaves" : 31, "learning_rate" : 0.05,
           "bagging_fraction" : 0.7, "bagging_seed" : 0,
           "num_threads" : 4, "colsample_bytree" : 0.7 }
```

Also, the least value of test **MAE** obtained was using **LightGBM** which was **0.02925782343434** and the test **R2 score** was **0.9874584758374857**

The baseline model I used was linear Regression. Comparing the LightGBM results with the baseline model, I can observe that the test MAE has decreased considerably, as for Linear Regression, the MAE value is approximately 0.049.

I have reported my leaderboard score in the Kaggle competition. This result was obtained on the offline test data which was provided by Kaggle. The trained model was used to test on the testing data.

200	new	Deepika Kanade		0.0245	1
-----	-----	----------------	---	--------	---

According to my leaderboard position, I am ranked 200 which according to me is a decent rank. I obtained a MAE of 0.0245 on the offline test dataset provided by Kaggle. A lot of people participating in Kaggle competition focus more on feature engineering and obtain good results by understanding features the best. Due to limited time and knowledge regarding this, I couldn't spend more time on feature engineering.

Also, after observing some kernels of the people at top on the leaderboard, I observed that they had used Neural Networks in their system. Given the huge dataset Neural Network would have definitely proved effective but would have required a lot of resources to run. So, with limited resources and some amount of feature engineering, I think this is a decent rank.

Interpretation of results:

After doing some initial analysis and observing the performance on different models on this dataset, I concluded the following:

- The dataset has non-linearities in it. When a simple Linear regression model is used for the dataset, the MAE is considerably high, but when a non-linear model such as Random forest or LightGBM is used, the MAE is very low.
- As the number of features are low, feature engineering plays a very important role. I tried reducing feature dimension by finding correlations between features, but it did not help. Reducing the size of the dataset by grouping and increasing features in the dataset proved to be effective.
- The most surprising thing was that even a basic linear regression model with all the features gave a decent R2 score, indicating that underfitting can be prevalent. So, to deal with that using of non-linear models and decreasing dataset was necessary.
- As a group in match had same final placement in that match, in spite of having different feature values, confused the model. Hence after grouping the data points, the output results were much better.

5. Summary and Conclusions

In this project, a variety of machine learning algorithms and models were experimented. As I have mentioned earlier, I found that the algorithm which works best for this dataset is where grouping of data points is done, and feature dimensions is increased by adding more features from this grouping and also some manual features. Also, LightGBM being fast and efficient for large datasets works the best.

However, this project is not perfect and there is a scope of improvement. This project can be further extended by 1) trying multiple Neural network models and hyper-tuning their parameters to get the best output 2) to do more amount of feature engineering to include some more features in the system.

6. References

- [1] Kaggle dataset - <https://www.kaggle.com/c/pubg-finish-placement-prediction/data>
- [2] Linear Regression - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- [3] Random Forest Regressor - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [4] EE 660 Lecture notes and Discussion notes
- [5] www.wikipedia.com
- [6] Light GBM - <https://lightgbm.readthedocs.io/en/latest/Python-Intro.html>
- [7] XGBoost - https://xgboost.readthedocs.io/en/latest/python/python_intro.html
- [8] Preprocessing - <http://scikit-learn.org/stable/modules/preprocessing.html>
- [9] Kaggle discussions and kernels - <https://www.kaggle.com/c/pubg-finish-placement-prediction/discussion>